Project 1c. Use gpg to create PGP key, sign and verify software.

The learning goals of this assignment is to

- learn how to clone an image with the new working version of gpg software, and the related crypto libraries, TLS network protocol modules, software that manages X.509 certificates and CMS data.
- learn how to create your own public/private key using gpg.
- publish your own public key on gpg server and your web server on Amazon Linux 2 instance.
- retrieve a public key from a gpg key server and import into your key ring.
- learn how to sign a software and proper way to post software archive and related signature on a web server.
- learn how to verify the open source software with PGP signature.

The assignment will be graded by

1. whether the public key is properly posted in the learner's web server,
2. whether the httpd.bz2 software is properly signed, and the signature and software are properly posted for others to verify,
3. whether the learner has demonstrated proper verification of an opensource software package.

Assignment Topic:

Use gpg to generate your public/private keys, publish your public key, download and verify the authenticity and integrity of open source software packages, sign a software package and post them properly.

GnuPG is a complete and free implementation of the OpenPGP standard as defined by **RFC4880** (also known as *PGP*). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories. GnuPG, also known as *GPG*, is a command line tool with features for easy integration with other applications. A wealth of **frontend applications** and **libraries** are available [GnuPG][1].

Since the version of GnuPG software on the current (as of 8/8/2022) Amazon Linux 2 image was an old GnuPG 2.0.22 version, it does not recognize the newer encryption key algorithm used in signing the software and does not implement newer protocols for communicating with key servers. To carry out our exercises, it is necessary to upgrade your instance by installing the new version of GnuPG software 2.3.7 (take about 40 min) or to clone a new instance from an image I have prepared recent with new GnuPG 2.3.7 software (take about 5 min). Due to lack of references, it takes me two weeks to finally figure out the steps and related to parameters in the commands. Even though the compiling gpg 2.3.7 will take about 40 min, but you will get a working up-to-date software that realize the current PGP standards, along the way you learn the scope of the designing and developing, and the related installation of a key cybersecurity software package. For learners with less time, just clone

[1] GnuPG: https://gnupg.org/index.html web site.

In the session below we download the most recent version of GnuPG software, version 2.3.7 (gnupg-2.3.7.tar.bz2) from GnuPG download web site https://gnupg.org/download/index.html and try to verify its signature (gnupg-2.3.7.tar.bz2.sig) is properly signed by the release key of the organization.

In the following, you will be asked to login to your instance and carry out the same sequence of commands, with your input in foreground red and background yellow. We will highlight the key facts/results with colors. Please pay attention to them. Along the way, you will be asked to save some browser images of the intermediate results as the deliverables for this project.

TIPS: Save the sessions of your project as text files. They contain rich information spit out by the gpg command. You can review them offline and learn from them.

Here is a session which shows the execution of gpg commands on an instance with gpg 2.0.22 software. It shows gpg cannot recognize the public key algorithm.

```
[ec2-user@ip-172-31-88-234 ~]$ gpg -h
gpg (GnuPG) 2.0.22
libgcrypt 1.5.3
Copyright (C) 2013 Free Software Foundation, Inc.
….
[ec2-user@ip-172-31-88-234 ~]$ mkdir pgp; cd pgp
[ec2-user@ip-172-31-88-234 pgp]$ curl -O https://gnupg.org/ftp/gcrypt/gnupg/gnupg-2.3.7.tar.bz2
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 7421k  100 7421k    0     0  4876k      0  0:00:01  0:00:01 --:--:-- 4876k
[ec2-user@ip-172-31-88-234 pgp]$ curl -O https://gnupg.org/ftp/gcrypt/gnupg/gnupg-2.3.7.tar.bz2.sig
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   119  100   119    0     0    305      0 --:--:-- --:--:-- --:--:--   305
[ec2-user@ip-172-31-88-234 pgp]$ gpg --verify gnupg-2.3.7.tar.bz2.sig gnupg-2.3.7.tar.bz2
gpg: directory `/home/ec2-user/.gnupg' created
gpg: new configuration file `/home/ec2-user/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/ec2-user/.gnupg/gpg.conf' are not yet active during this run
gpg: keyring `/home/ec2-user/.gnupg/pubring.gpg' created
gpg: Signature made Mon 11 Jul 2022 11:37:31 AM UTC using ? key ID 905BA208
gpg: Can't check signature: Invalid public key algorithm
```

Here is a session which shows what a newer gpg 2.3.7 software will do in verifying this software package. We use an instance where we have compiled and installed new gpg2.3.7 software and related libraries.

```
[root@ip-172-30-1-197 pgp]# gpg --verify gnupg-2.3.7.tar.bz2.sig gnupg-2.3.7.tar.bz2
gpg: Signature made Mon 11 Jul 2022 11:37:31 AM UTC
gpg:                using ECDSA key 02F38DFF731FF97CB039A1DA549E695E905BA208
gpg: Good signature from "GnuPG.com (Release Signing Key 2021)" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 02F3 8DFF 731F F97C B039  A1DA 549E 695E 905B A208
```

Note that in the above command we use the same GnuPG version 2.3.7 software to check the integrity of its software archive with the signing signature created by its designer. How do we know the gpg we run is truly generated by the software from GnuPG.com? This is so called chicken-egg problem. How do we know the gpg software we run is authentic? This is called the ***trust bootstrapping problem***. One answer is to use another crypto software perhaps OpenSSL on a "trustworthy" machine Mac, Windows, or a Redhat Linux to verify the authenticity and integrity of the gnupg-2.3.7.tar.bz2 software archive. By "trustworthy" we mean the machine and related crypto and networking software are diligently checked and maintained by qualified system staffs. This is a good topic for a forum discussion.

Setup instructions:

Step 1a. Option 1. Clone an instance from an image with gpg 2.3.7. Estimated about 2 minutes.

Choose this option if you are time pressed, or not interested in learning the composition of an important cybersecurity software package and how to compile and install it.

Follow the steps in project1aV9.pdf to clone an instance with gpg 2.3.7 already installed.

Next, jump to Step 2 in Page 11.

Step 1b. Option 2. Install gpg 2.3.7 software on an instance cloned in project1a before 8/12/2022. Estimated about 40 minutes.

Choose this option if you are interested in learning the composition of an important cybersecurity software package and how to compile and install it.

Before you begin, you will need to login to your instance following the steps in Project 1a. Create a pgp directory using "mkdir pgp". We will download and compile software in this directory. Switch to the pgp directory with "cd pgp".

Step 1b.1. Install software development tools

Since our Amazon Linux 2 instance was not originally designed for developing software, the typical software development tools such as gcc, libtool, rcs, etc were not installed. We install them with yum groupinstall "Development Tools" command. Precede it with sudo to elevate the needed root privilege.

```
[ec2-user@ip-172-31-88-234 ~]$ sudo yum groupinstall "Development Tools"
It is recommended to switch to root for installing the gpg libraries and gpg command.
[ec2-user@ip-172-31-88-234 ~]$ mkdir pgp
[ec2-user@ip-172-31-88-234 ~]$ cd pgp
[ec2-user@ip-172-31-88-234 pgp]$ sudo bash
[root@ip-172-31-88-234 pgp]#
Note that the shell prompt will change to #
```

Step 1b.2. Install newer version of GnuPG software.

The GnuPG download site shows the following software packages, required libraries/tools.
https://gnupg.org/download/index.html

Name	Version	Date	Size	Tarball	Signature
GnuPG	2.3.7	2022-07-11	7421k	download	download
GnuPG (LTS)	2.2.36	2022-07-06	7103k	download	download
Libgpg-error	1.45	2022-04-07	992k	download	download
Libgcrypt	1.10.1	2022-03-28	3689k	download	download
Libgcrypt (LTS)	1.8.9	2022-02-07	2918k	download	download
Libksba	1.6.0	2021-06-10	646k	download	download
Libassuan	2.5.5	2021-03-22	558k	download	download
ntbTLS	0.3.1	2022-04-07	343k	download	download
nPth	1.6	2018-07-16	293k	download	download
Pinentry	1.2.0	2021-08-25	486k	download	download

Note that for compiling GnuPG 2.2.36, we need to compile the related Libgcrypt (LTS) 1.8.9 not the 1.10.1. The Tarball column contains the actual source archive while the Signture column contains the signature of the related software archive, signed with the release key of GnuPG. The related release keys can be found in **https://gnupg.org/signature_key.asc** and the integrity check step can be found in https://gnupg.org/download/integrity_check.html

Before we can install GnuPG version 2.3.7, we need to install the following ibraries in the order listed.

npth (https://gnupg.org/ftp/gcrypt/npth/)
libgpg-error (https://gnupg.org/ftp/gcrypt/libgpg-error/)
libgcrypt (https://gnupg.org/ftp/gcrypt/libgcrypt/)
libksba (https://gnupg.org/ftp/gcrypt/libksba/)
libassuan (https://gnupg.org/ftp/gcrypt/libassuan/)
ntbtls (https://gnupg.org/ftp/gcrypt/ntbtls/ntbtls-0.3.1.tar.bz2)

Without ntbtls installed, a support software called dirmngr will not be installed and gpg –keyserver keyserver.ubuntu.com -recv-key … type of command will not work. Then we cannot download the public keys from the keyserver for verification with the gpg command and need to visit keyserver web site and manually download the text version of the public key. Ntbtls is a client-only TLS implementation developed by Werner Koch, the original designer of GnuPG. It has a smaller footprint than GnuTLS which requires more libraries and is a full implementation of TLS.

For each of the required libraries, we go through the following typical 4-step installation process:

```
configure      # run the configure command to consult related software installed and update makefiles
make           # actually compile the software modules and generate the libraries or the executables
make check     # test whether the software generated passed the test suite developed by the developers
make install   # actually install the software to the desired locations for future references or linkage
```

In the following, we include the related commands with detailed options for compiling the libraries and gpg software. You need to switch to root user with "sudo bash" before executing these commands.

Good instructions for compiling gnupg-2.3.7 can be found in

https://www.linuxfromscratch.org/blfs/view/svn/postlfs/gnupg.html

Here we adapt some of the same installation steps.

Step 1b.2.1 Setup linking path and install text2htm texinfo for related gpg documentation

To facilitate the linking of those related libraries, we create a pgp.conf with single line content "/usr/lib" to the /etc/ld.so.conf.d directory. We also add /usr/lib to the LD_LIBRARY_PATH

```
[root@ip-172-31-88-234 pgp]# echo /usr/lib > /etc/ld.so.conf.d/pgp.conf
[root@ip-172-31-88-234 pgp]# export LD_LIBRARY_PATH=/usr/lib:${LD_LIBRARY_PATH}
```

In the installation of these libraries, we use the option --prefix=/usr as configure option for installing libraries to /usr/lib. Since the later installed libraries often depends on the earlier installed libraries such as libgpg-error, or libgcrypt, the above two commands help set up the environment variable for the configure and make commands to look for proper directories to test the existence of some dependent libraries and link them.

```
[root@ip-172-31-88-234 pgp]# yum install text2html texinfo
```

Text2html and texinfo is used to generate the related GnuPG documents.

Step 1b.2.2. Install npth library.

The NPth package contains a very portable POSIX/ANSI-C based library for Unix platforms which provides non-preemptive priority-based scheduling for multiple threads of execution (multithreading)

inside event-driven applications. All threads run in the same address space of the server application, but each thread has its own individual program-counter, run-time stack, signal mask and errno variable.

Run the following commands:

```
cd /home/ec2-user/pgp
curl -O https://gnupg.org/ftp/gcrypt/npth/npth-1.6.tar.bz2
tar xjf npth-1.6.tar.bz2
cd npth-1.6
./configure --prefix=/usr
make
make check
make install
```

Note that it is important to include --prefix=/usr option so that the library will be installed in the chosen location, /usr/lib, for others in the system to link.

Step 1b.2.3. Install libgpg-error library

Libgpg-error is a small library that originally defined common error values for all GnuPG components. Among these are GPG, GPGSM, GPGME, GPG-Agent, libgcrypt, Libksba, DirMngr, Pinentry, SCdaemon.

Run the following commands:

```
cd /home/ec2-user/pgp
curl -O https://gnupg.org/ftp/gcrypt/libgpg-error/libgpg-error-1.45.tar.bz2
tar xjf libgpg-error-1.45.tar.bz2
cd libgpg-error-1.45
./configure --prefix=/usr
make
make check
make install
install -v -m644 -D README /usr/share/doc/libgpg-error-1.45/README
```

Step 1b.2.4. Install libgcrypt library

"Libgcrypt is a general purpose cryptographic library originally based on code from GnuPG. It provides functions for all cryptograhic building blocks: symmetric cipher algorithms (AES, Arcfour, Blowfish, **Camellia**, CAST5, ChaCha20 DES, GOST28147, Salsa20, **SEED**, Serpent, Twofish) and modes (ECB,CFB,CBC,OFB,CTR,CCM,GCM,OCB,POLY1305,AESWRAP), hash algorithms (MD2, MD4, MD5, GOST R 34.11, RIPE-MD160, SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256, TIGER-192, Whirlpool), MACs (HMAC for all hash algorithms, CMAC for all cipher algorithms, GMAC-AES, GMAC-CAMELLIA, GMAC-TWOFISH, GMAC-SERPENT, GMAC-SEED, Poly1305, Poly1305-AES,

Poly1305-CAMELLIA, Poly1305-TWOFISH, Poly1305-SERPENT, Poly1305-SEED), public key algorithms (RSA, Elgamal, DSA, ECDSA, EdDSA, ECDH), large integer functions, random numbers and a lot of supporting functions.

Libgcrypt works on most POSIX systems and many pre-POSIX systems. It can also be built using a cross-compiler system for Microsoft Windows.

See also its **Wikipedia** entry."

The above is quoted from https://gnupg.org/software/libgcrypt/index.html

This is an example of a crypto library implementation. It is worth while to observe the compilation/test of encryption, authentication, integrity checking, and hashing methods needed for a crypto library. In "make check" step, please observe various tests and time measurements for these critical crypto methods. The other common used crypto library is OpenSSL.

Run the following commands:

```
cd /home/ec2-user/pgp
curl -O https://gnupg.org/ftp/gcrypt/libgcrypt/libgcrypt-1.10.1.tar.bz2
tar xjf libgcrypt-1.10.1.tar.bz2
cd libgcrypt-1.10.1/
./configure --prefix=/usr
make
make -C doc html
makeinfo --html --no-split -o doc/gcrypt_nochunks.html doc/gcrypt.texi
makeinfo --plaintext       -o doc/gcrypt.txt         doc/gcrypt.texi
make check
make install
install -v -dm755   /usr/share/doc/libgcrypt-1.10.1 &&
install -v -m644    README doc/{README.apichanges,fips*,libgcrypt*} \
                    /usr/share/doc/libgcrypt-1.10.1

install -v -dm755   /usr/share/doc/libgcrypt-1.10.1/html &&
install -v -m644 doc/gcrypt.html/* \
                    /usr/share/doc/libgcrypt-1.10.1/html &&
install -v -m644 doc/gcrypt_nochunks.html \
                    /usr/share/doc/libgcrypt-1.10.1     &&
install -v -m644 doc/gcrypt.{txt,texi} \
                    /usr/share/doc/libgcrypt-1.10.1
```

Step 1b.2.5. Install Libksba

Libksba is a library to make the tasks of working with X.509 certificates, CMS data and related objects more easy. It provides a highlevel interface to the implemented protocols and presents the data in a

consistent way. There is no more need to worry about all the nasty details of the protocols. The API gives the C programmer an easy way of interacting with the data. It copes with the version details X.509 protocols tend to have as well as with the many different versions and dialects. Applications must usually cope with all of this and it has to be coded over and over again. *Libksba* hides this by providing just one API which does the Right Thing™. Support for new features will be added as needed.

Run the following commands:

```
cd /home/ec2-user/pgp
curl -O https://gnupg.org/ftp/gcrypt/libksba/libksba-1.6.0.tar.bz2
tar xjf  libksba-1.6.0.tar.bz2
cd libksba-1.6.0
./configure --prefix=/usr
make
make check
make install
```

Step 1b.2.6. Install Libassuan

Libassuan is a small library implementing the so-called *Assuan protocol*. This protocol is used for IPC between most newer GnuPG components. Both, server and client side functions are provided.

Run the following commands:

```
cd /home/ec2-user/pgp
curl -O https://gnupg.org/ftp/gcrypt/libassuan/libassuan-2.5.5.tar.bz2
tar xjf  libassuan-2.5.5.tar.bz2
cd libassuan-2.5.5
./configure --prefix=/usr
make
make -C doc html
makeinfo --html --no-split -o doc/assuan_nochunks.html doc/assuan.texi
makeinfo --plaintext       -o doc/assuan.txt          doc/assuan.texi
make check
make install
install -v -dm755   /usr/share/doc/libassuan-2.5.5/html &&
install -v -m644 doc/assuan.html/* \
                 /usr/share/doc/libassuan-2.5.5/html &&
install -v -m644 doc/assuan_nochunks.html \
                 /usr/share/doc/libassuan-2.5.5      &&
install -v -m644 doc/assuan.{txt,texi} \
                 /usr/share/doc/libassuan-2.5.5
```

Step 1b.2.7. Install ntbtls

ntbTLS is a tiny TLS 1.2 only implementation designed to be used with Libgcrypt and LibKSBA. In particular, this library has no certificate verification code - this need to be done by the caller. For example the GnuPG component *dirmngr* already has code to verify certificates (for CRL and OCSP checking) and thus *ntbTLS* is a good fit for accessing objects over the network.

Run the following commands:

```
cd /home/ec2-user/pgp
curl -O https://www.gnupg.org/ftp/gcrypt/ntbtls/ntbtls-0.3.1.tar.bz2

tar xjf  ntbtls-0.3.1.tar.bz2
cd ntbtls-0.3.1
./configure --prefix=/usr
make
make check
make install
```

Step 1b.2.8. Install GnuPG 2.3.7

GnuPG is a command line tool without any graphical user interface. It is a universal crypto engine which can be used directly from a command line prompt, from shell scripts, or from other programs. Therefore GnuPG is often used as the actual crypto backend of other applications.

Run the following commands:

```
cd /home/ec2-user/pgp
curl -O https://www.gnupg.org/ftp/gcrypt/gnupg/gnupg-2.3.7.tar.bz2
tar xjf gnupg-2.3.7.tar.bz2
cd gnupg-2.3.7
./configure --prefix=/usr
make
make check
make install
install -v -m755 -d /usr/share/doc/gnupg-2.3.7/html          &&
install -v -m644    doc/gnupg_nochunks.html \
                    /usr/share/doc/gnupg-2.3.7/html/gnupg.html &&
install -v -m644    doc/*.texi doc/gnupg.txt \
                    /usr/share/doc/gnupg-2.3.7 &&
install -v -m644    doc/gnupg.html/* \
                    /usr/share/doc/gnupg-2.3.7/html
```

Finally let us set the LD_LIBRARY_PATH environment variable value with value /usr/lib in /etc/profile so that gpg can access the shared libraries: libgcrypt.so.20 by any user in the system

```
sudo echo export LD_LIBRARY_PATH=/usr/lib >> /etc/profile.d/sh.local
```

You can verify whether LD_LIBRARY_PATH by executing "echo $LD_LIBRARY_PATH" command next time you login.

```
[ec2-user@ip-172-31-89-183 ~]$ echo $LD_LIBRARY_PATH

/usr/lib
```

Now let us verify the gpg-2.3.7.tar.bz2 software with gpg --verify command

Make sure in your pgp directory you have gnupg-2.3.7.tar.bz2.sig and gnupg-2.3.7.tar.bz2 files.

If not, you can download them with curl -O command.

```
[root@ip-172-31-89-183 pgp]# curl -O https://gnupg.org/ftp/gcrypt/gnupg/gnupg-2.3.7.tar.bz2.sig
[root@ip-172-31-89-183 pgp]# curl -O https://gnupg.org/ftp/gcrypt/gnupg/gnupg-2.3.7.tar.bz2
[root@ip-172-31-89-183 pgp]# gpg --verify gnupg-2.3.7.tar.bz2.sig gnupg-2.3.7.tar.bz2
gpg: directory '/root/.gnupg' created
gpg: keybox '/root/.gnupg/pubring.kbx' created
gpg: Signature made Mon 11 Jul 2022 11:37:31 AM UTC
gpg:                using ECDSA key 02F38DFF731FF97CB039A1DA549E695E905BA208
gpg: Can't check signature: No public key
```

We can download the gnupg signing keys and import to your key ring with the following two commands. You can refer to the web page https://www.gnupg.org/download/integrity_check.html for more info on gnupg software integrity check.

```
[root@ip-172-31-89-183 pgp]# curl -O https://gnupg.org/signature_key.asc
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5365  100  5365    0     0  13409      0 --:--:-- --:--:-- --:--:-- 13379
[root@ip-172-31-89-183 pgp]# gpg --import signature_key.asc
gpg: key BCEF7E294B092E28: 1 signature not checked due to a missing key
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key BCEF7E294B092E28: public key "Andre Heinecke (Release Signing Key)" imported
gpg: key 528897B826403ADA: 4 signatures not checked due to missing keys
gpg: key 528897B826403ADA: public key "Werner Koch (dist signing 2020)" imported
gpg: key E98E9B2D19C6C8BD: 2 signatures not checked due to missing keys
gpg: key E98E9B2D19C6C8BD: public key "Niibe Yutaka (GnuPG Release Key)" imported
gpg: key 549E695E905BA208: 1 signature not checked due to a missing key
prigpg: key 549E695E905BA208: public key "GnuPG.com (Release Signing Key 2021)" imported
gpg: Total number processed: 4
gpg:               imported: 4
gpg: no ultimately trusted keys found
```

With the GnuPG signing public key in place, let us verify the GnuPG software again.

```
[root@ip-172-31-89-183 pgp]# gpg --verify gnupg-2.3.7.tar.bz2.sig gnupg-2.3.7.tar.bz2
gpg: Signature made Mon 11 Jul 2022 11:37:31 AM UTC
gpg:                using ECDSA key 02F38DFF731FF97CB039A1DA549E695E905BA208
gpg: Good signature from "GnuPG.com (Release Signing Key 2021)" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 02F3 8DFF 731F F97C B039  A1DA 549E 695E 905B A208
```

Now it reports the software is signed by GnuPG.com and also the ID indicates that it is the release signing key for 2021.

Step 2. Generate the public/private key pair and publish the public key.

Now logout from the instance and remote log back in as ec2-user.

Step 2.1. Create your own pgp public key pair.

cd pgp

Run the command "gpg --full-generate-key"

It will show the version of gpg software that is installed in your instance.

If this is the first time you run the command, it will create a .pgp in your home directory and create public keyring and private keyring there to save the created or imported keys.

It will then ask you to specify the type of key you like to create. We will select 1 which is RSA and RSA algorithm. We will choose 4096 bits as key size, instead of default 3072 bits and choose 0 for "key does not expire". Confirm that by entering

Gpg then asks you to enter the USER_ID information. Enter your full name as Real Name, email address you used in your Coursera account as Email address, and "`Test Account for Coursera CS5910" as Comment. Enter O for "Okay".`

Gpg then ask you to enter the passphrase to protect your private key. Make sure you remember it.

The following is the session data shown when we execute this command. Make sure to replace with your own User_ID info.

```
[ec2-user@ip-172-31-89-183 ~]$ gpg --full-generate-key
gpg (GnuPG) 2.3.7; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/home/ec2-user/.gnupg' created
```

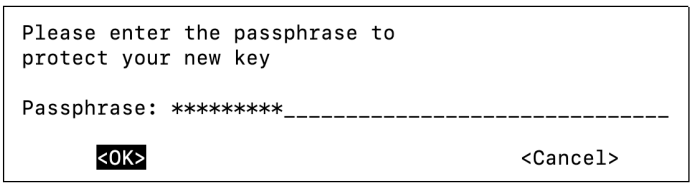
```
gpg: keybox '/home/ec2-user/.gnupg/pubring.kbx' created
Please select what kind of key you want:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (sign only)
  (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Edward Chow
Email address: edwardchowc@gmail.com
Comment: Test Account for Coursera CS5910
You selected this USER-ID:
    "Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

You got the prompt asking you to enter the passphrase to protect your public and private keys.



After enter the passphrase, you enter tab key to highlight <OK>, then hit enter.
Make sure you remember or write down this passphrase, since you will be asked again later to enter passphrase when opening the private key to encrypt files.

We need to generate a lot of random bytes. It is a good idea to perform

some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.
gpg: /home/ec2-user/.gnupg/trustdb.gpg: trustdb created
gpg: directory '/home/ec2-user/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/ec2-user/.gnupg/openpgp-revocs.d/618BA201742473DDE3CC70310CEE63348D9BE2F9.rev'
public and secret key created and signed.

```
pub   rsa4096 2022-07-30 [SC]
      618BA201742473DDE3CC70310CEE63348D9BE2F9
uid                   Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.com>
sub   rsa4096 2022-07-30 [E]
```

618BA201742473DDE3CC70310CEE63348D9BE2F9 is the ***key ID*** for the PGP public key generated. We can use it in submitting the public key to a key server for sharing with others. The last few hexadecimal digits, normally we use the last 8, e.g., here it is 8D9BE2F9, for the uniqueness as ***partial key ID*** to search for public keys on the keyserver portals. Some portal often requires us to precede the partial key ID with 0x to explicit indicating it is a hexadecimal digits.

Step 2.2. Publish your public key.
You can

1) Publish your public key on your web server as a web page.
2) Attach your public key as attachment or main text in your email. The recipient can import them to their public key ring.
3) Publish your public key to pgp key servers.

Step 2.2.1. Create a public key file for posting on your own web site or on PGP open public key servers.
To publish public key as a web page or text email attachment, first we need to extract the public key and output them in ASCII readable format. We run the following command:

```
[ec2-user@ip-172-31-89-183 ~]$ gpg --armor --export edwardchowc@gmail.com > mypubkey.asc
```

where --armor option generates ASCII instead of binary file.
--export option extracts the related public key out from the public keyring. We can use KeyID (portion of key fingerprint) or email address as option value.
We redirect the result on console to save as a file with .asc (which is file extension for GPG public key in ASCII format; the binary file will have .pgp as file extension).

Step 2.2.2. Post public key file on your web site.

Let us create a pgp directory on our web site and copy the mypuykey.asc there.

[ec2-user@ip-172-31-89-183 ~]$ mkdir /var/www/html/pgp; cp mypubkey.asc /var/www/html/pgp

If you see permission denied message, add "sudo" before your command.
Your ec2-user account should have the write privilege on /var/www/html since we have modified the access rights of that directory to include ec2-user during LAMP server package setup during project1a.
You can verify the web access by typing the following url into your browser.

https://<your instance IP address>/pgp/mypubkey.asc

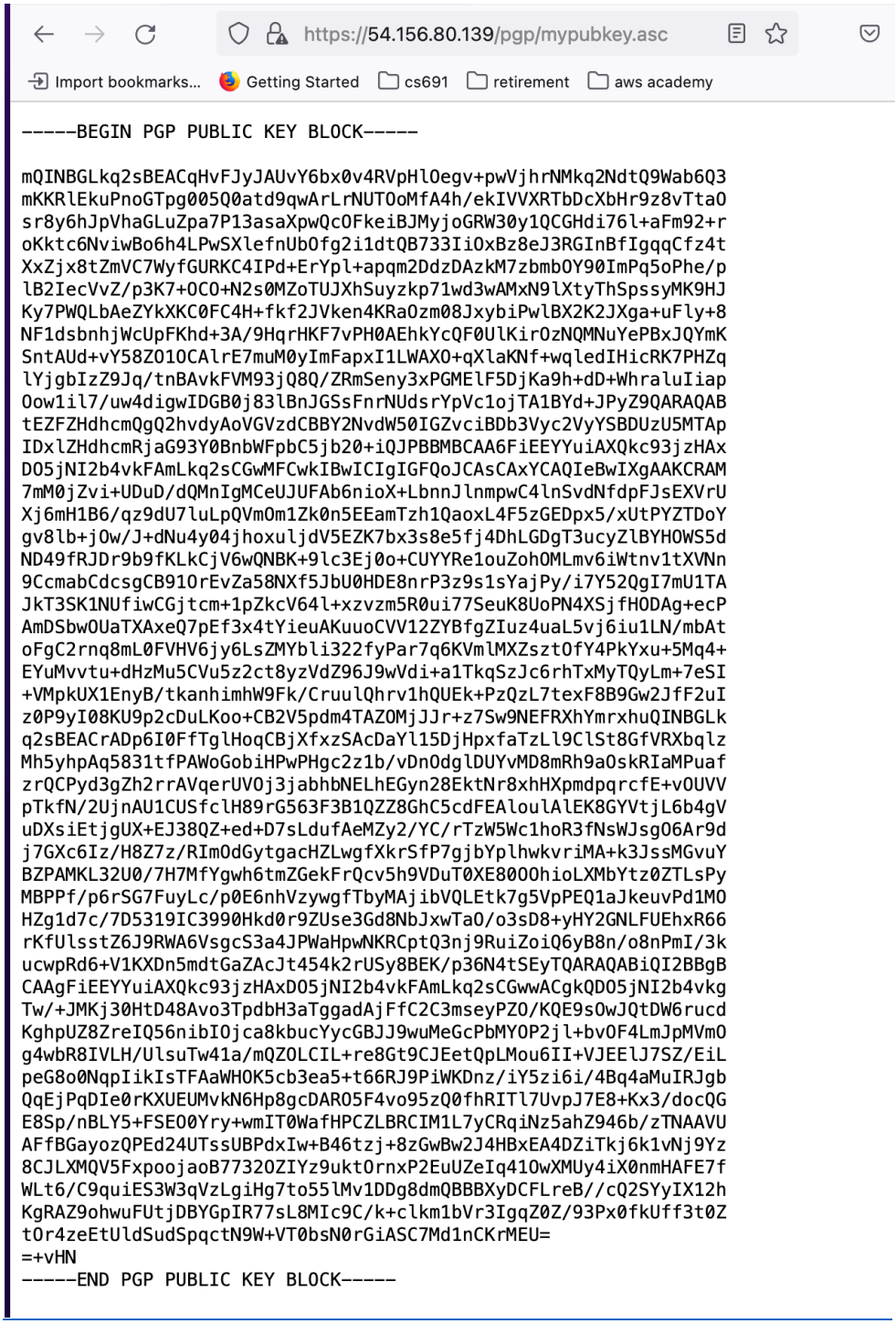


Figure 1. Your PGP public key on your web server.

Try to use Firefox browser, since Google Chrome is set to block the access of web site where the server certificate is self-signed or not signed by the known list of CA in its Security Database.

You can now tell your friend to retrieve your public key from the httpd web server of your instance, if you add their local IP address to the instance's source IP address list in the security group firewall rules. See the procedure in Project1aV8.pdf.

How others can use your public key?
They can download this mypubkey.asc file and then use "gpg –import mypubkey.asc" file to their key ring for future verification of your signed messages/documents and encryption of messsages sent just for you.

Project Deliverable #1. Capture the browser image showing your public key similar to Figure 1.

The image is the first deliverable of the project1c you should submit.

Step 2.2.3. Publish the public key to a pgp key server

Run the following command:

```
[ec2-user@ip-172-31-89-183 ~]$ gpg --keyserver keyserver.ubuntu.com --send-key
618BA201742473DDE3CC70310CEE63348D9BE2F9
gpg: sending key 0CEE63348D9BE2F9 to hkp://keyserver.ubuntu.com
```

To retrieve a public key, you can use the gpg command with --search-key option. The gpg can search for the key server preconfigured or you can specify a specific pgp key server you trust or close-by.

```
[ec2-user@ip-172-31-89-183 ~]$ gpg --keyserver keyserver.ubuntu.com --search-keys edwardchowc@gmail.com
gpg: data source: http://162.213.33.8:11371
(1)     Edward Chow C (Edward Chow Testing account for Coursera Education) <ed
          4096 bit RSA key 880C6C3FC9F99E80, created: 2017-07-28
(2)     C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.c
          4096 bit RSA key 0209F7F3E1C41764, created: 2022-01-24
(3)     Edward Chow C (This is a test account for CS4910/CS5910) <edwardchowc@
          4096 bit RSA key C545FC94CF484155, created: 2020-09-25
(4)     C. Edward Chow (Test Account) <edwardchowc@gmail.com>
          4096 bit RSA key 644B0D1ABA8D03B8, created: 2020-12-11
(5)     Shubhanjay Varma (Test Account for Coursera CS5910) <edwardchowc@gmail
          4096 bit RSA key BD68375DADE5D63B, created: 2020-05-26
(6)     C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.c
          4096 bit RSA key 790ABE30EDFD024B, created: 2020-10-08
(7)     C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.c
          4096 bit RSA key 131C89792D36DC4B, created: 2017-07-29
(8)     C. Edward Chow (Test Account for Coursera) <edwardchowc@gmail.com>
          4096 bit RSA key DA9C7D6260C1217B, created: 2020-12-11
(9)     4096 bit RSA key D325C390B46F4EAD, created: 2018-09-18
(10)    Shubhanjay Varma (Test Account for Coursera CS5910) <edwardchowc@gmail
          4096 bit RSA key 2958AAD423B3E9EE, created: 2020-05-26
Keys 1-10 of 10 for "edwardchowc@gmail.com".  Enter number(s), N)ext, or Q)uit > N
```

Note that in the above, we did not see my new pgp key with edwardchowc@gmail.com created today 2022-07-30. It is posted that fast. After about 5 minutes later, the same command show my new public key in key #5.

```
[ec2-user@ip-172-31-89-183 ~]$ gpg --keyserver keyserver.ubuntu.com --search-keys edwardchowc@gmail.com
gpg: data source: http://162.213.33.8:11371
(1)     Edward Chow C (Edward Chow Testing account for Coursera Education) <ed
          4096 bit RSA key 880C6C3FC9F99E80, created: 2017-07-28
(2)     C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.c
          4096 bit RSA key 0209F7F3E1C41764, created: 2022-01-24
(3)     Edward Chow C (This is a test account for CS4910/CS5910) <edwardchowc@
          4096 bit RSA key C545FC94CF484155, created: 2020-09-25
(4)     C. Edward Chow (Test Account) <edwardchowc@gmail.com>
          4096 bit RSA key 644B0D1ABA8D03B8, created: 2020-12-11
(5)     Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.com>
          4096 bit RSA key 0CEE63348D9BE2F9, created: 2022-07-30
(6)     Shubhanjay Varma (Test Account for Coursera CS5910) <edwardchowc@gmail
          4096 bit RSA key BD68375DADE5D63B, created: 2020-05-26
(7)     C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.c
          4096 bit RSA key 790ABE30EDFD024B, created: 2020-10-08
(8)     C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.c
          4096 bit RSA key 131C89792D36DC4B, created: 2017-07-29
(9)     C. Edward Chow (Test Account for Coursera) <edwardchowc@gmail.com>
          4096 bit RSA key DA9C7D6260C1217B, created: 2020-12-11
(10)      4096 bit RSA key D325C390B46F4EAD, created: 2018-09-18
(11)    Shubhanjay Varma (Test Account for Coursera CS5910) <edwardchowc@gmail
          4096 bit RSA key 2958AAD423B3E9EE, created: 2020-05-26
Keys 1-11 of 11 for "edwardchowc@gmail.com".  Enter number(s), N)ext, or Q)uit > 5
gpg: key 0CEE63348D9BE2F9: "Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.com>" not
changed
gpg: Total number processed: 1
gpg:          unchanged: 1
```

Vulnerability of PGP old key server: One key observations on the list here: others can abuse PGP free posting service by posting fake public key with email addresses not own by them, such as those posted keys 6 and 11 are from your remote classmates. Potentially they can disguise me w.r.t. edwardchowc@gmail.com. The old pgp key servers have this vulnerability since their protocol implements do not verify the ownership of the email address before posting them into the pgp key database.

Note that the pgp key servers were used to be synchronized, once key was published on one key server, it would be soon available on all pgp key servers. But three key issues cause the operation problem,

1) Some pgp key servers are ***taken down due to not compliant with EU GDPR regulation***,
2) The ***misuse or abuse and privacy issues*** that plague old SKS Keyservers, and
3) ***6/2019 certificate spamming attack*** against two high-profile contributors in the OpenPGP community and took down quite a few pgp key servers. Anyone who attempts to import a poisoned

certificate into a vulnerable OpenPGP installation will very likely break their installation in hard-to-debug ways. https://gist.github.com/rjhansen/67ab921ffb4084c865b3618d6955275f

For example, once the most popular pgp key server site https://sks-keyservers.net/ posts msg:

"This service is deprecated. This means it is no longer maintained, and new HKPS certificates will not be issued. Service reliability should not be expected.

Update 2021-06-21: Due to even more GDPR takedown requests, the DNS records for the pool will no longer be provided at all."

Two new types of pgp key servers emerge. One requires verification of the ownership of the email address; the other does not verify the ownership of the email, but implement more security checks against the cyber attacks.

Type 1. They require the verification of the email address before your ID information is released. These include the use of keys.openpgp.org. "However this keyserver only includes User IDs for keys whose owners have personally confirmed via email (basically eliminating large swaths of of the PGP ecosystem). It also does not include *any* 3rd party signatures on keys to mitigate the possibility of a "poisoned key" attack. As of December 2021, this is the default (if none is configured by the user) keyserver for GnuPG packaged by Debian since gnupg2 2.2.17-1 (released in 2019)."

First download your pgp key from your instance to your local machine with scp commands.

```
cchow@MacBook-Pro privateKey % scp -i cchow_awsac_cs5910_pkey.pem ec2-user@54.156.80.139:mypubkey.asc .
mypubkey.asc                  100% 3183    15.6KB/s   00:00
cchow@MacBook-Pro privateKey % mv mypubkey.asc edwardchowcPubkey.asc
```

Replace the private key and public key files in above commands with those of yours.

Step 1. Enter http://keys.openpgp.org/ in a browser. Then hit load button link. It will bring you to a web page for choosing and upload your PGP public key.

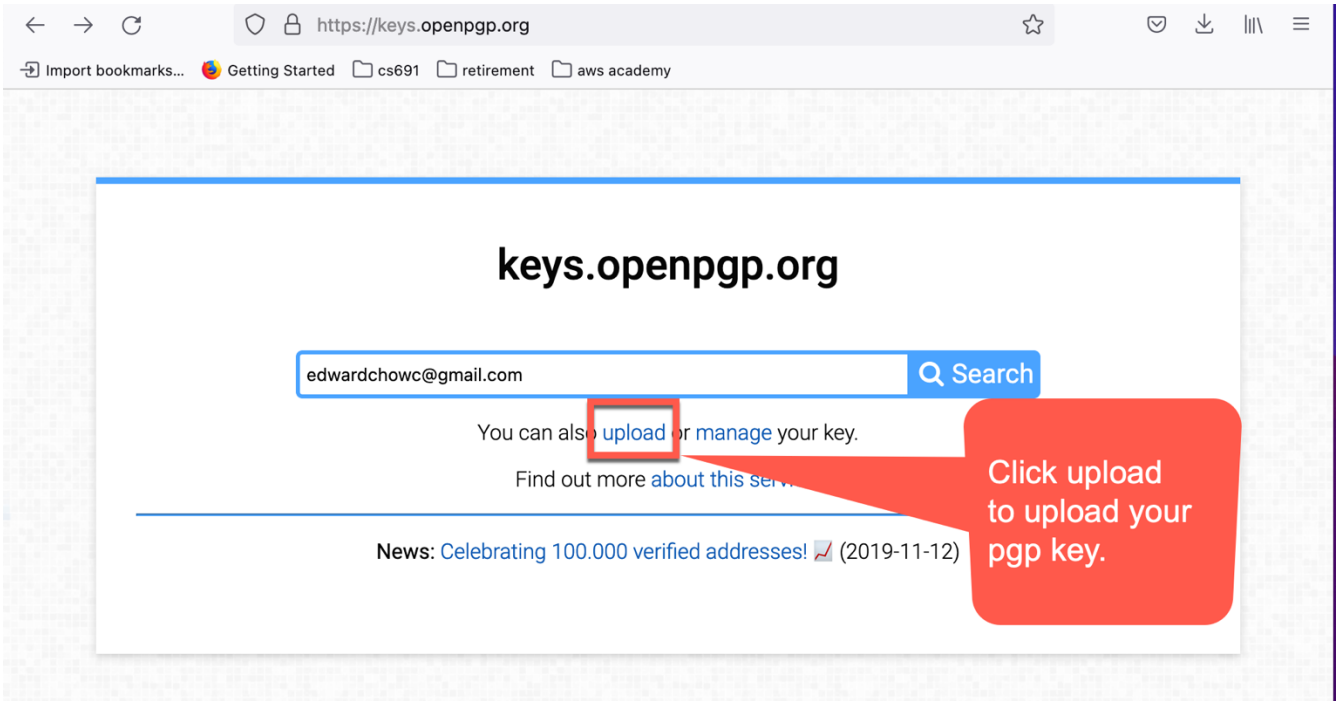


Figure 2. keys.openpgp.org main web page with upload and manage key button.

Note that by clicking on the manage button, you will visit the manage key web page where you enter your email and an email will be sent to you. See Figure 3. It gives you opportunity to remove some of PGP public key in the search list. This is a newly added feature demanded by GDPR regulation and does not exist in the older pgp key server site.

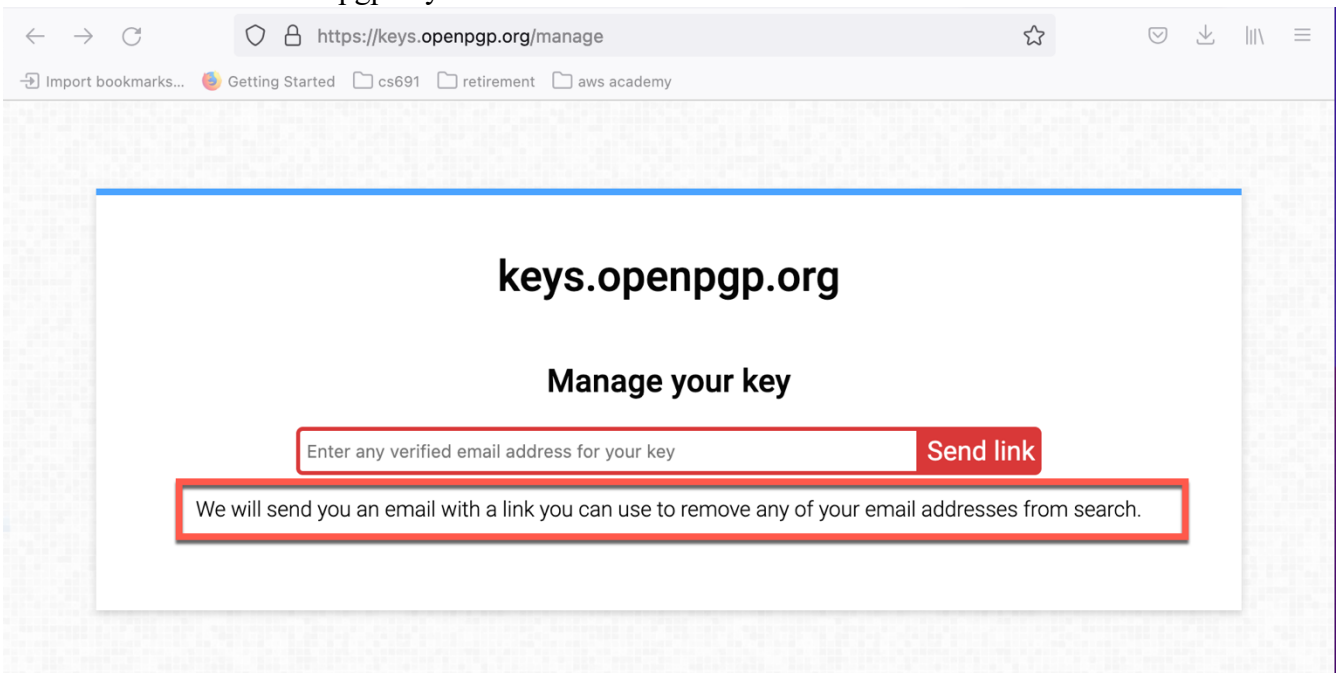


Figure 3. keys.openpgp.org Manage key web page.

For example, after you select your pgp key file in .asc file format and click Upload button with the web page.

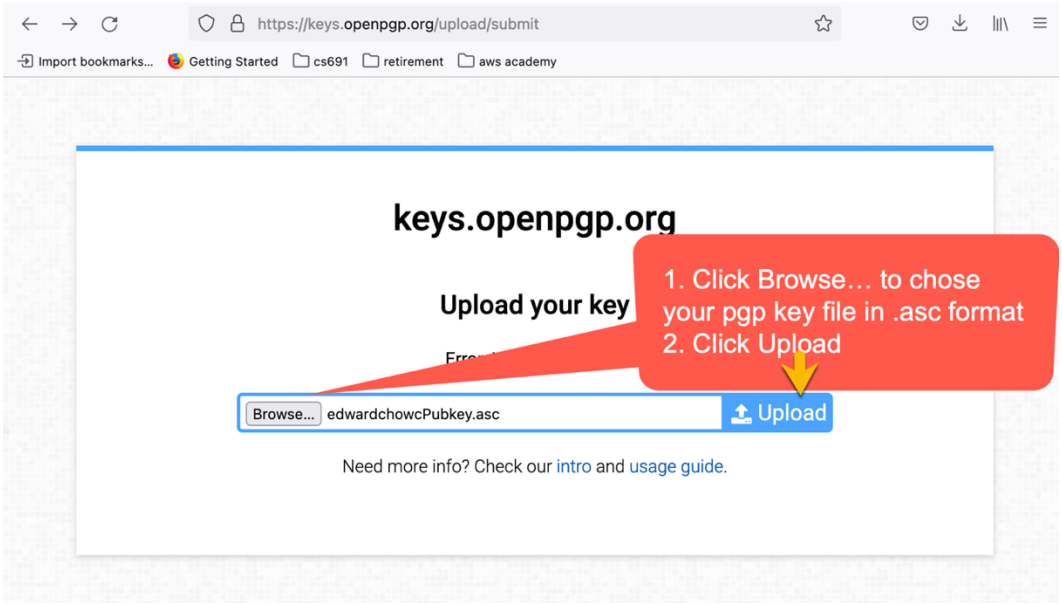


Figure 4. keys.openpgp.org Upload key web page.

You receive this response web page

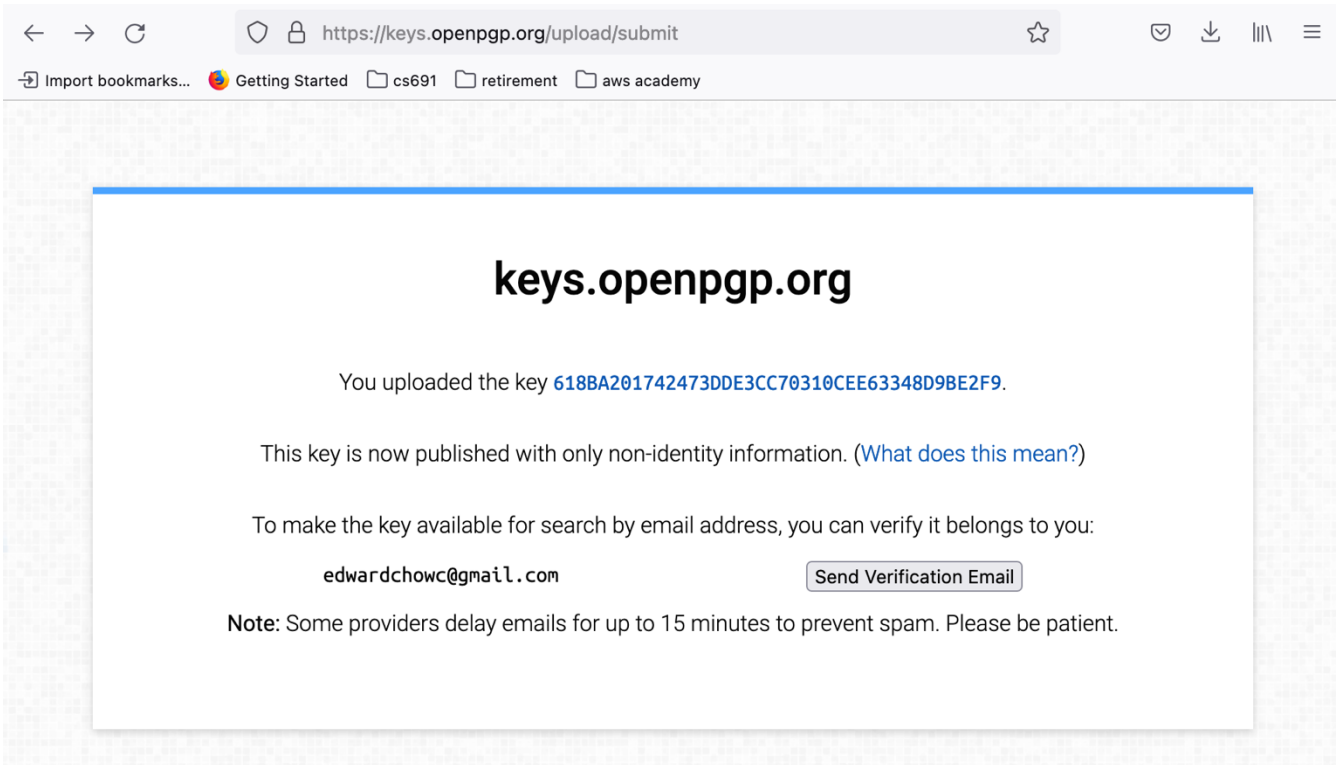


Figure 5. Request to send Verification email.

Note that the web page says “This key is now published with only non-identity information.” Basically on this site at this moment you can only search with key ID and download the related public key. It will not provide you with the related identity information such as the email address associated with this key.

Click “Send Verification Email”

You will receive an email to verify your pgp key upload request similar to the one below:

From: keyserver@keys.openpgp.org <keyserver@keys.openpgp.org>
Date: Saturday, July 30, 2022 at 6:42 PM
To: edwardchowc@gmail.com <edwardchowc@gmail.com>
Subject: Verify edwardchowc@gmail.com for your key on keys.openpgp.org

Hi,

This is an automated message from keys.openpgp.org. If you didn't request this message, please ignore it.

OpenPGP key: 618BA201742473DDE3CC70310CEE63348D9BE2F9

To let others find this key from your email address "edwardchowc@gmail.com", please click the link below:

https://keys.openpgp.org/verify/iT5EcpThXxYnaVo30jqvuVfkJDL6HJK2VjsWztwWQDf

You can find more info at keys.openpgp.org/about.

https://keys.openpgp.org
distributing OpenPGP keys since 2019

After click on the link in the email and enter the edwardchowc@gmail.com on query box of https://keys.openpgp.org/. See Figure 2.

We got the following response web page. Click the link in the middle, the browser will download a copy of the PGP public key.

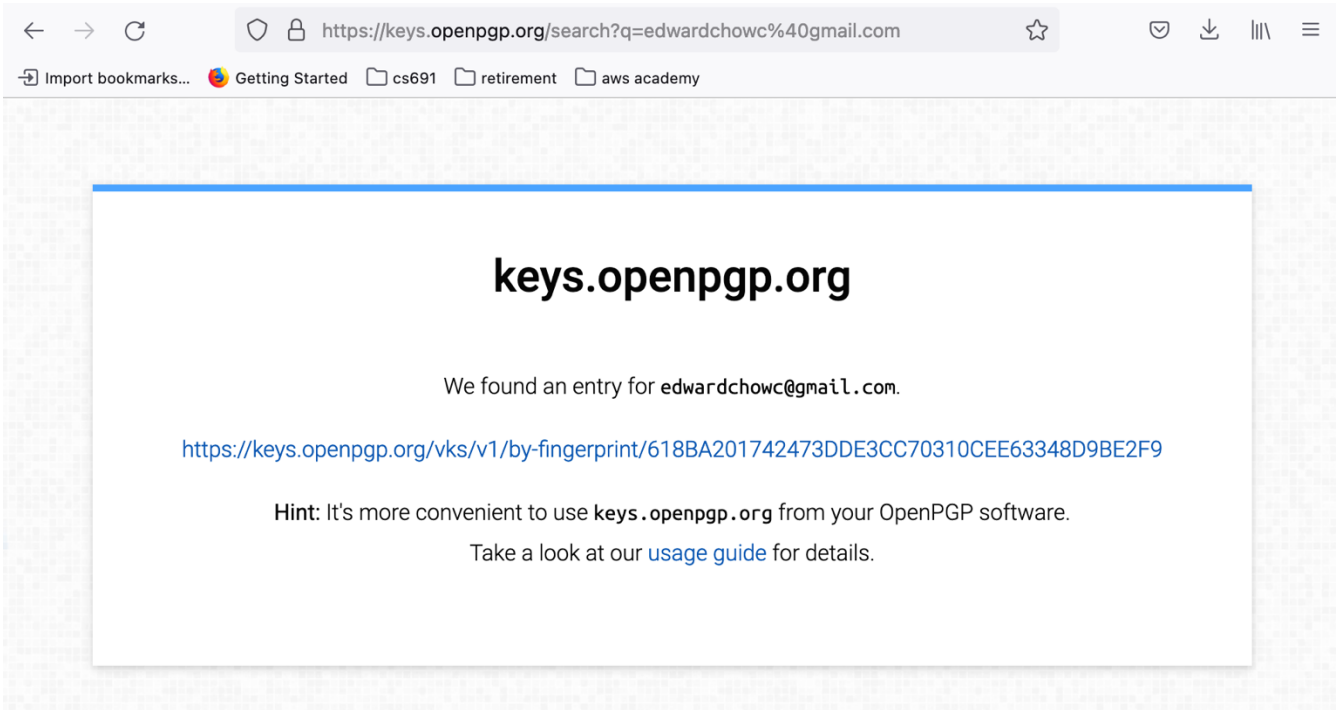


Figure 6. Search result of type 1 PGP public key posted on keys.openpgp.org.

Project Deliverable #2. Capture the browser image showing your public key similar to Figure 6.

The image is the second deliverable of the project1c you should submit.

Examples of key servers:

- keys.openpgp.org based on hagrid and verifying
- keys.mailvelope.com based on Mailvelope and verifying

Type 2. There are keyservers like keyserver.ubuntu.com which does not require the verification of email address so far but may require such verification in the future. Note that the 3rd party signatures which endorse the key will be stripped off. It helps reduce the severity of spam attack and re-establish the trust.

Example:

- keyserver.ubuntu.com

You are encouraged to try post your pgp key on both types.

You can also enter https://keyserver.ubuntu.com/ or any pgp key server with GUI. Enter search string and get the public key on the web browser. You can then copy and save the pub key as .asc file and import into your key ring.

Here are steps:

First download your pgp key from your instance to your local machine with scp command. Replay the name with your own.

```
cchow@MacBook-Pro privateKey % scp -i cchow_awsac_cs5910_pkey.pem ec2-user@54.156.80.139:mypubkey.asc .
mypubkey.asc                  100% 3183    15.6KB/s   00:00
cchow@MacBook-Pro privateKey % mv mypubkey.asc edwardchowcPubkey.asc
```

Replace the private key and public key files in above commands with those of yours.

Step. Search your PGP key on keyserver.ubuntu.com

Note that we have previously at Page 14 upload our pgp public key with

```
gpg --keyserver keyserver.ubuntu.com --send-key 618BA201742473DDE3CC70310CEE63348D9BE2F9
```

a. Type in your email in the text box and then click Search Key button.

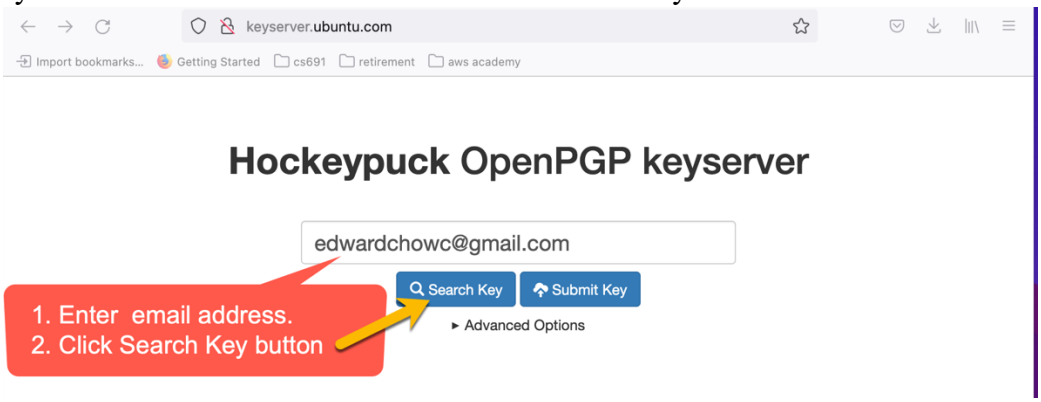


Figure 7. Search PGP key on keyserver.ubuntu.com

b. Search result.

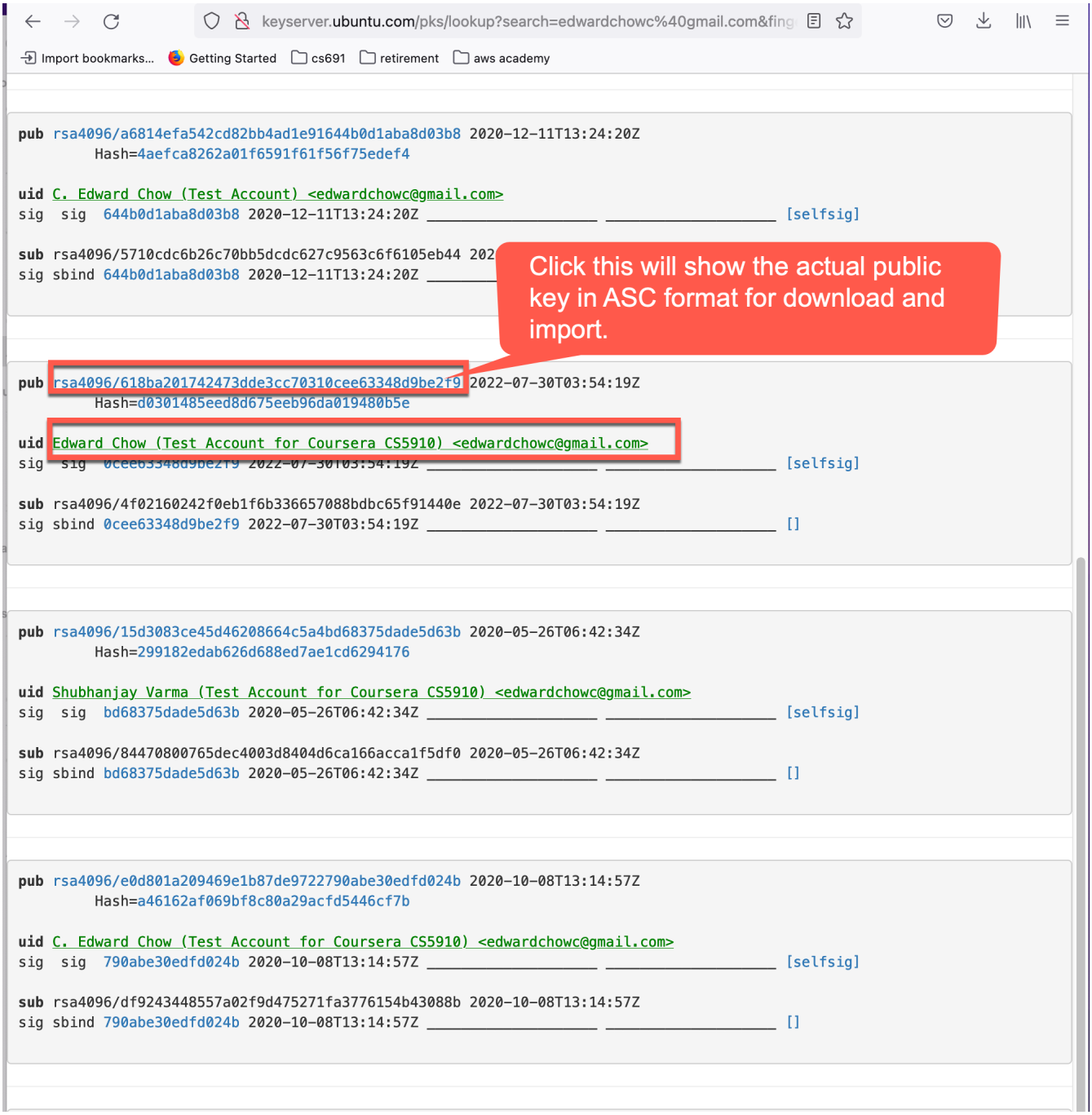


Figure 8. Search result. The last few bytes and the creation day of the key identifies it.

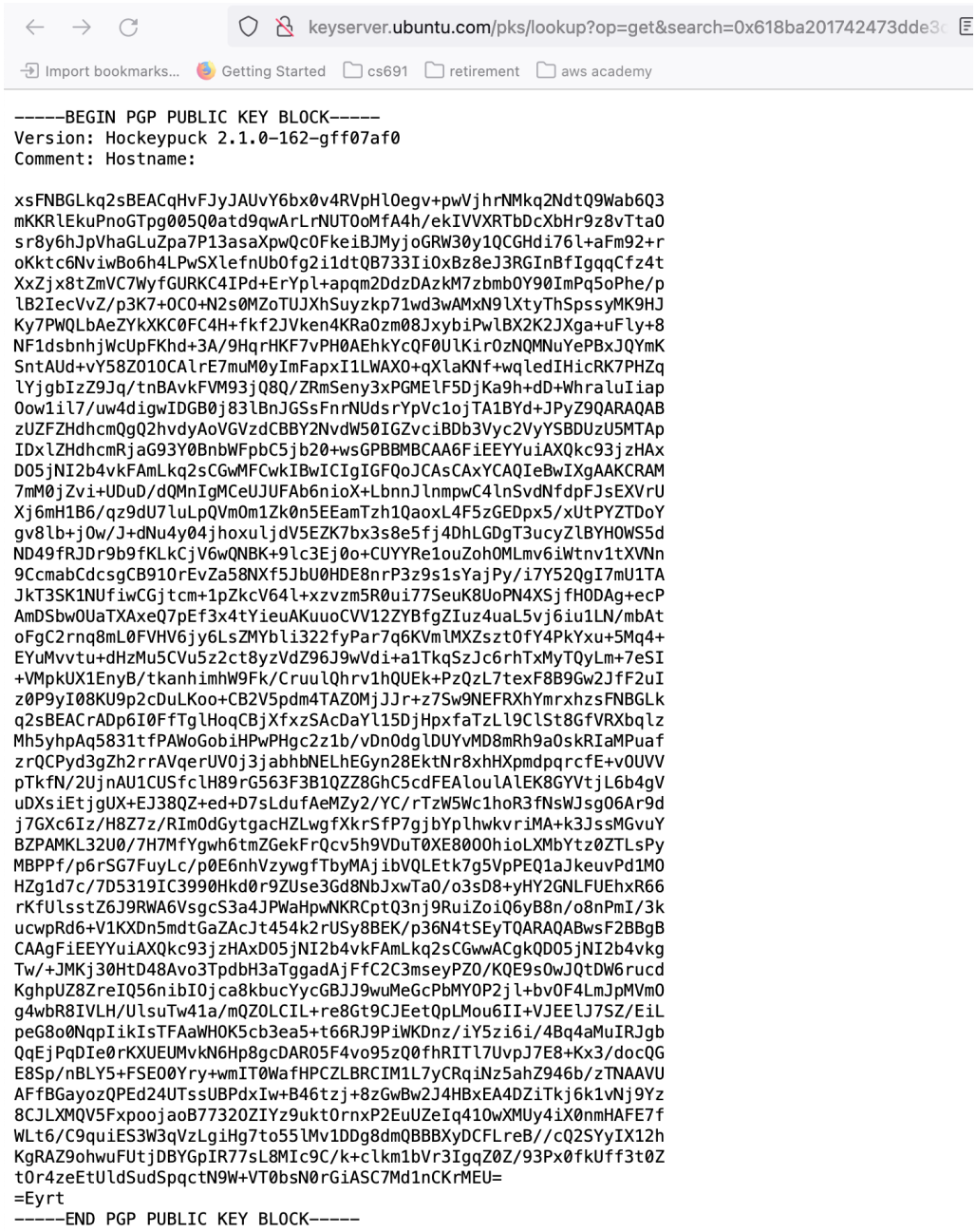


Figure 9. One's PGP public key on keyserver.ubuntu.com.

Project Deliverable #3. Capture the browser image showing your public key similar to Figure 9.

The image is the third deliverable of the project1c you should submit.

Step 3. Verify Opensource Software Packages.

In this exercise, we show an example of verifying software package which utilizes PGP signing and integrity checking. The software package is the popular Apache httpd software package. The current release is 2.4.54. You are encouraged to download even more recent version. We will use curl commands to download the source code of Apache httpd source code and related key files. The –O option save the file with the same file name of the url. Then use gpg --verify to verify the authenticity and integrity of the download source. Apache has a lot of projects. All their code signing public keys are consolidated in a single KEYS file and make it easier for the user to retrieve. They should include the link to this KEYS file in the specific software download pages.

Execute the list of commands showing in the following session:

```
[ec2-user@ip-172-31-89-183 ~]$ mkdir /home/ec2-user/proj1c; cd /home/ec2-user/proj1c
[ec2-user@ip-172-31-89-183 proj1c]$ curl -O http://archive.apache.org/dist/httpd/KEYS
[ec2-user@ip-172-31-89-183 proj1c]$ curl -O http://archive.apache.org/dist/httpd/httpd-2.4.54.tar.bz2
[ec2-user@ip-172-31-89-183 proj1c]$ curl -O http://archive.apache.org/dist/httpd/httpd-2.4.54.tar.bz2.asc
[ec2-user@ip-172-31-89-183 proj1c]$ ls -al
total 7808
drwxrwxr-x 2 ec2-user ec2-user      78 Jul 31 15:36 .
drwx------ 7 ec2-user ec2-user     184 Jul 31 15:26 ..
-rw-rw-r-- 1 ec2-user ec2-user 7434530 Jul 31 15:36 httpd-2.4.54.tar.bz2
-rw-rw-r-- 1 ec2-user ec2-user     874 Jul 31 15:36 httpd-2.4.54.tar.bz2.asc
-rw-rw-r-- 1 ec2-user ec2-user  549907 Jul 31 15:35 KEYS
[ec2-user@ip-172-31-89-183 proj1c]$ gpg --import KEYS
....
gpg: key 19B033D1760C227B: public key "Christophe JAILLET <christophe.jaillet@wanadoo.fr>" imported
gpg: key D377C9E7D1944C66: public key "Stefan Eissing (icing) <stefan@eissing.org>" imported
gpg: Total number processed: 69
gpg:    skipped PGP-2 keys: 27
gpg:            imported: 41
gpg:           unchanged: 1
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
[ec2-user@ip-172-31-89-183 proj1c]$ gpg --verify httpd-2.4.54.tar.bz2.asc httpd-2.4.54.tar.bz2
gpg: Signature made Mon 06 Jun 2022 02:23:05 PM UTC
gpg:              using RSA key 26F51EF9A82F4ACB43F1903ED377C9E7D1944C66
gpg: Good signature from "Stefan Eissing (icing) <stefan@eissing.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:         There is no indication that the signature belongs to the owner.
Primary key fingerprint: 26F5 1EF9 A82F 4ACB 43F1  903E D377 C9E7 D194 4C66
```

Key observations:

1. httpd-2.4.54.tar.bz2.asc is much shorter, 874 bytes, vs. that of the software archive, httpd-2.4.54.tar.bz2, 7,434,530 bytes.
2. The first parameter for the “gpg --verify” command is the signature, not the software archive.

Save the above gpg –verify execution session results in a web page called httpdVerified.html by wrapping the text session data with <pre></pre> tag.

```
<h1>httpd software verification results</h1>
<pre>

--- put the httpd gpg verification text session data here ---

</pre>
```

copy it to your pgp web site with the following command.

```
[ec2-user@ip-172-31-89-183 proj1c]$ cp httpdVerified.html /var/www/html/pgp
```

Figure 10 shows the browser image of the httpdVerified.html web page on my instance.

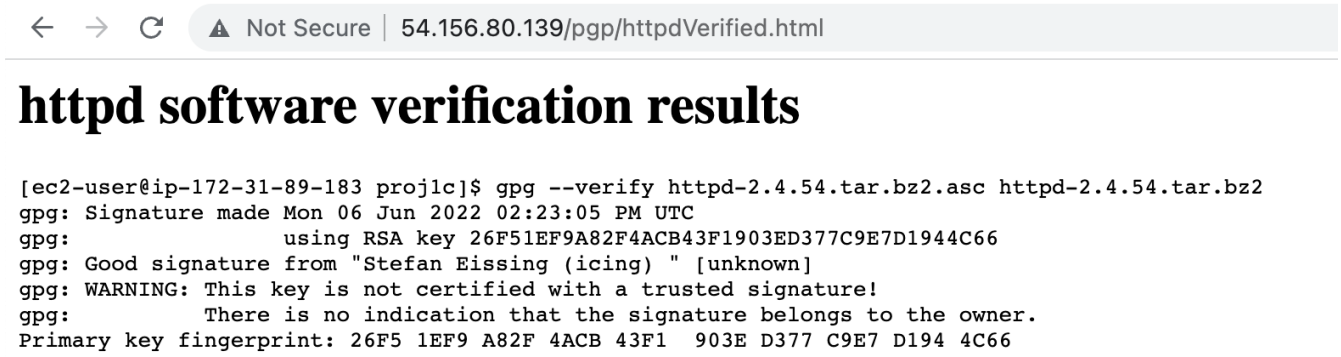


Figure 10. GnuPG verification results of httpd software archive.

Project Deliverable #4. Capture the browser image of the httpdVerified.html served by your instance and submit it as the 4th deliverable of your project 1c.

Step 3. Sign a software with your private key and post it properly.

In this exercise we pretend that we just create a software archive and we will learn how to post it properly on our web site, together with its signature key, and our singing public key for others to download and perform integrity check.

In the following sequence of commands, we first rename the httpd-2.4.25.tar.bz2 in your project1c directory as httpd.bz2 and pretend it is a software archive we just developed and ready for distribution. Then we use “gpg --detach-sig” command with option --output to produce the signature of httpd.bz2 and save it as httpd.bz2.gpg. Finally we copy the software archive (httpd.bz2) and its signature

(httpd.bz2.gpg) to pgp web directory for other to download. Note that the signing public key is mypubkey.asc which has been already copied to /var/www/html/pgp during a step in Step 1 Page 13 earlier.

```
[ec2-user@ip-172-31-89-183 ~]$ cd /home/ec2-user/proj1c
[ec2-user@ip-172-31-89-183 ~]$ cp httpd-2.4.54.tar.bz2  httpd.bz2
[ec2-user@ip-172-31-89-183 ~]$ gpg --output httpd.bz2.gpg --detach-sig httpd.bz2
```

You will prompt to enter the passphrase for protecting the private key so that gpg can retrieve it from the security database for signing the software archive. This is done in Step 3.1 Page 12.

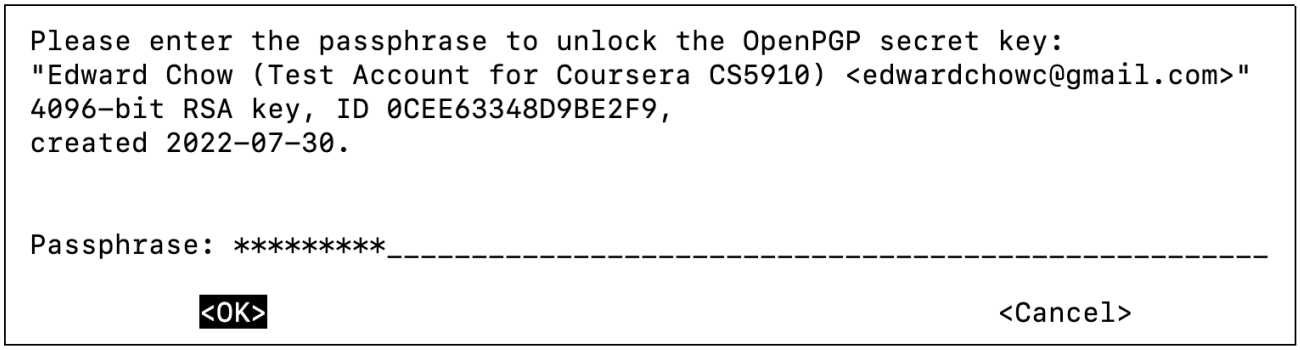


Figure 11. gpg asks for the passphrase to open the private key.

```
[ec2-user@ip-172-31-89-183 ~]$ ls -al
[ec2-user@ip-172-31-89-183 ~]$ cat httpd.bz2.gpg

[ec2-user@ip-172-31-89-183 ~]$ cp httpd.bz2 httpd.bz2.gpg  /var/www/html/pgp
```

Create a software download page called myhttpd.html with the following content

```
<h1> my signed httpd software package</h1>

Here is my software archive <a href="httpd.bz2">httpd.bz2</a>, its signature <a
href="httpd.bz2.gpg">signature</a>, and <a href="mypubkey.asc">my software release
public key</a>.

---Add a description on how gpg can be used to verify your httpd.bz2 software.---

---Add the gpg session with httpd.bz2.gpg is indeed signed by you  ---

cp myhttpd.html to /var/www/html/pgp
```

Here is the browser image of myhttpd.html served by my instance.

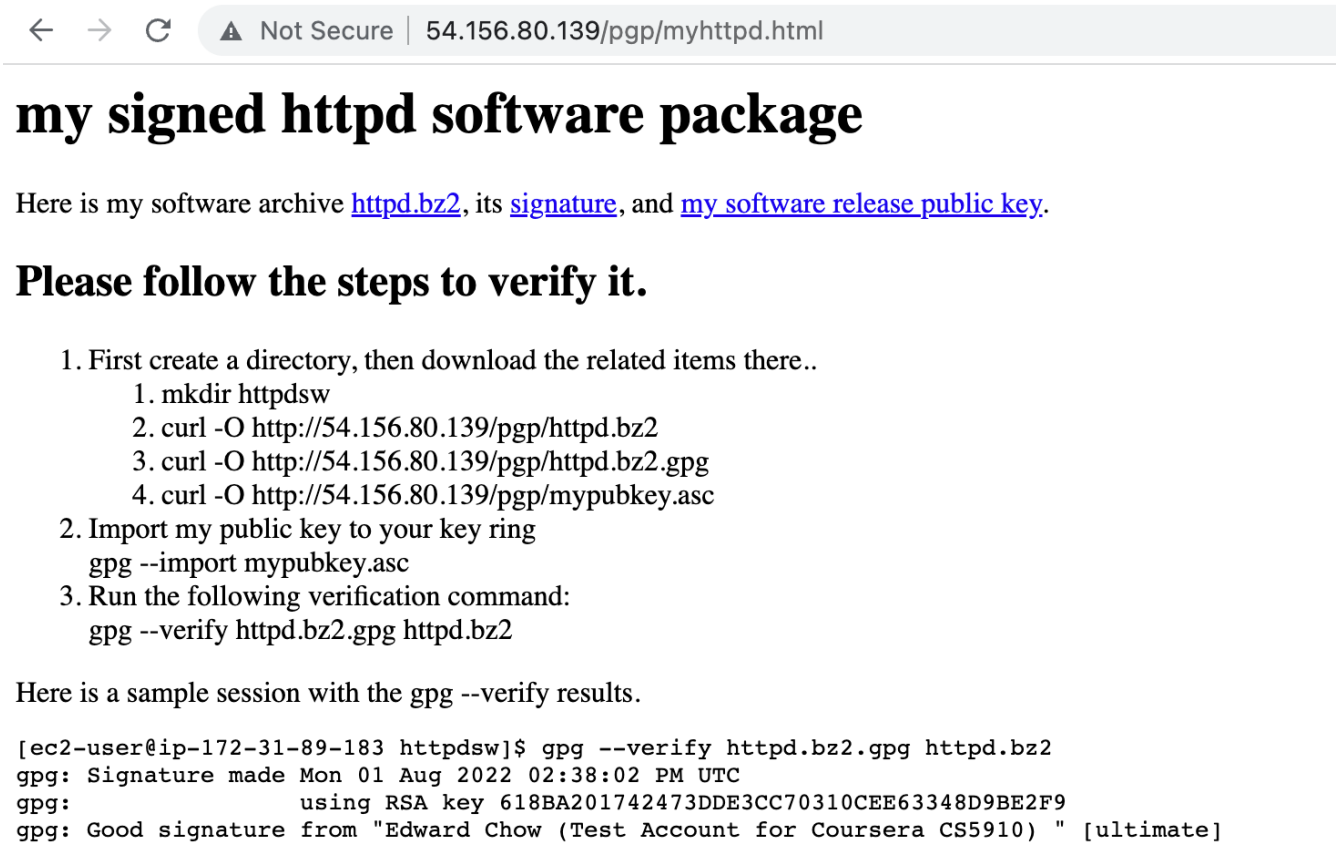


Figure 12. Sample of a proper software download web page.

Project Deliverable #5. Capture the browser image of the httpdVerified.html served by your instance similar to Figure 12 and submit it as the 5th deliverable of your project 1c.

You grade will be based on how concise and clear your description on how to verify your httpd.bz2 with proper gpg commands**.**

How to submit:

You will be required to submit your assignment with the browser images of

1. https://<yourInstancePublicIP>/pgp/mypubkey.asc
2. Search result of your PGP public key posted on keys.openpgp.org
3. Your PGP public key on keyserver.ubuntu.com
4. https://<yourInstancePublicIP>/pgp/puttyVerified.html
5. https://<yourInstancePublicIPAddress>/pgp/myhttpd.html

How to complete your assignment:

Follow the instructions in the project description to set up those web page answers.

Metrics for evaluation: Here are the five key questions you will be asked in peer reviewing of this project exercise.

Prompt 1. Does the user properly post the public key on his/her pgp web site?

Prompt 2. Does the image shows the proper search result on keys.openpgp.org?

Prompt 3. Does the image shows the proper search result on keyserver.ubuntu.com?

Prompt 4. Does the httpdVerified.html web page clearly shows the learner has verified that the putty software is from the original source and its integrity is not violated?

Prompt 5. Does the learner properly display all information necessary for others to download and verify the httpd.bz2 posted on the web site?